Distributed Applications Development

Computer Engineering	3rd Year	1st Semester	2020-21	Periodic Evaluation
Project	Limit date for results divulgation: 18 Jan 2022			
Date: 26 Oct 2021	**Delivery Date: 18 Dec 2021**			

Project – vCard

1. OBJECTIVE

This project's objective is to implement a Single Page Application (SPA) for the "vCard" platform, using Vue.js framework for the development of the web client, Laravel and Node.js for the backend (a Restful API with Laravel and a WebSocket server with Node.js).

2. VCARD

A vCard is a virtual debit card associated to one mobile phone – there is no physical card associated to a vCard. Each vCard is identified by the mobile phone number and each mobile phone can only be associated to one vCard.

The vCard has a balance (amount of money currently on the card) whose value cannot be negative because the vCard is a debit card (it is not a credit card). The vCard owner can make debit transactions (operations that decrease the vCard balance) to pay for their purchases, transfer money to other vCards or to external financial entities (e.g. Paypal account or Bank account). On the other hand, vCard owners cannot make direct credit transactions (operations that increase the vCard balance), because the vCard balance can only be increased through a transfer from another vCard or from an external entity. For both types of transactions (debit and credit), the type and reference of payment will identify where the money is transferred to or transferred from.

When a vCard makes a debit transaction or receives money through a credit transaction, the balance is updated automatically, and the transaction (credit or debit) is saved on the vCard transactions history – all transactions involving a vCard must be kept on that vCard transactions history.

Note that a transfer between 2 vCards includes a pair of transactions (2 transactions), a debit transaction from the source vCard and a credit transaction on the destination vCard.

Transactions registered on the vCard can also have a description (free text) and a classification by debit (expense) or credit (income) category. Both the description and classification are optional, and its usage and organization are dependent of the vCard owner. The vCard platform provides a set of suggested debit and credit categories, that will be copied as default categories when a new vCard is created. Afterwards, the vCard owner is free to organize the categories as he pleases (including removing or adding new categories).

3. VCARD PLATFORM

The vCard platform includes two applications to handle vCards: a mobile application that handles one associated vCard (associated to the phone number) and a Web application where we can handle any vCard – one at time (the authentication credentials include the phone number that identifies which vCard will be handled on that session).

Both applications can be used to perform debit transactions, to access the vCard transactions history, to edit and classify transactions' categories and to change the vCard profile and settings. Also, any user (anonymous user) can create a vCard associated to a specific phone number – new vCards are always created with balance = 0€.

In the Web application the vCard owner can also configure the credit (income) and debit (expense) categories and access statistical information about the vCard transactions.

The Web application will also include the platform administration (administrators' have distinct credentials that include the email and password) where it is possible to configure the default debit and credit categories and other platform settings, access the list of vCards, block and unblock vCards, change the debit limit of vCards, delete (with softdelete if necessary) vCards, and manually add credit transactions - register transfers from an external entity to a specific vCard.

The backend of the vCard platform will include a set of services (Restful API; Websocket server; MQTT server; etc…) that coordinates all transactions and support all features for both applications.

Note: Depending on the curricular units being evaluated, the implemented applications and the backend scope may differ. Also, it might be necessary to implement two different versions of the backend (a simplified and a full version) – in this case, the applications should support both versions of the backend (both backend versions should share part of the public interface).

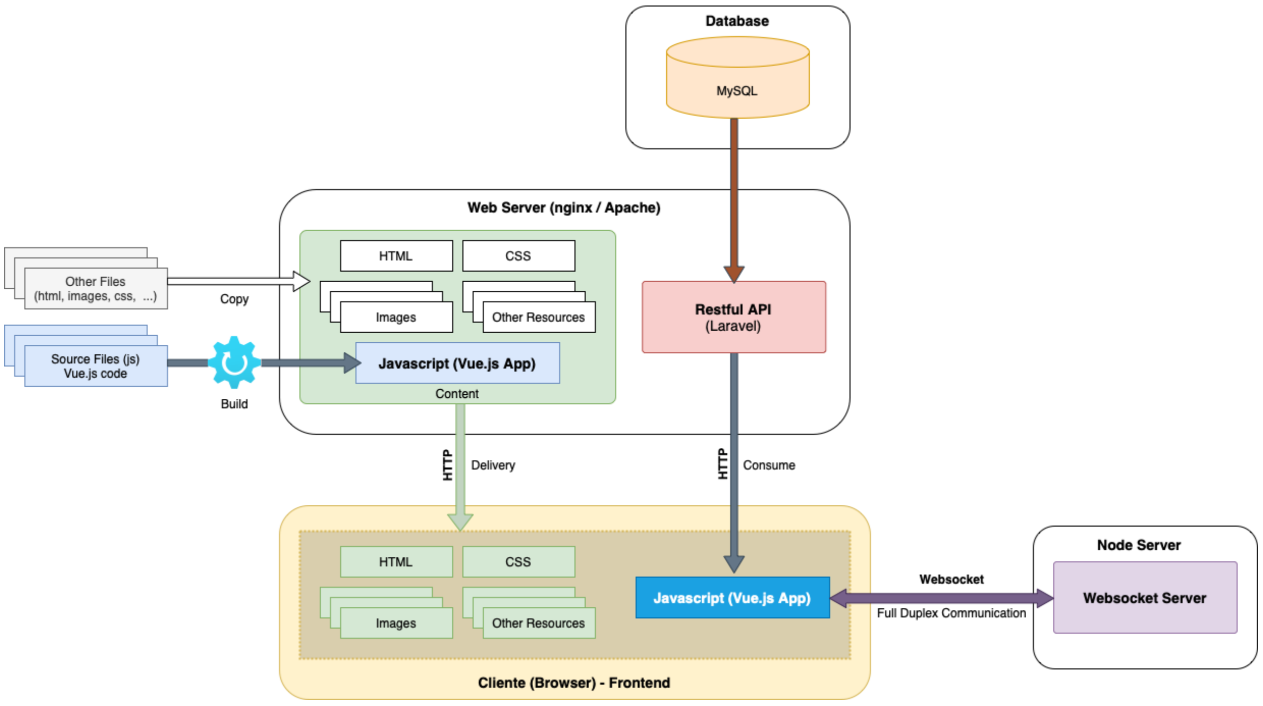
4. DAD PROJECT

The objective of the DAD project is to implement the Web application for the vCard platform. This web application will use the SPA application model and the Vue.js framework for the development of the web client. DAD project will also include a simplified version of the backend, with a MySQL database server, a Restful API implemented with Laravel and a Websocket server based on Node.js. The simplified version of the backend is mandatory (even if students implement a full version of the backend on a different curricular unit), ensuring that the DAD project can be deployed independently.

The simplified version of the backend will have all the services required for the web application to operate correctly, but all transactions with external entities (payments to external entities or payments from external entities) are simulated – the API will store the transactions on the database, but they will not be verified or integrated with other services.

4.1 ARCHITECTURE

The following diagram describes the architecture of the DAD project.



The client application source code will be partitioned in several Javascript source files (js), from which the webpack or similar tool will generate the application file (or files). The application file, which is also a Javascript file, and all the static assets (the html file, css files, images, etc.) are placed on the web server so that the application can be delivered through the Web – the application (JS file) is executed on the client (browser) but is delivered through a Web server.

The application on the client (browser) will consume the Restful API service, which is implemented in Laravel and will have several endpoints that provide data to the client or allow the client to perform operations. The Laravel Restful API will read and store data on a MySQL database.

The project also includes a Websocket server, implemented in Javascript and running on a Node.js server. A full-duplex communication channel is established between the client application and the Websocket server, which allows instant messaging between them. The Websocket server acts as an intermediary between several instances of the client application, so that is possible to establish full duplex communication between any instances of the client application (using the Websocket server as a communication intermediary).

It is also possible to use other libraries, technologies or databases that complement the described architecture. For instance, “NoSQL” databases or cache technologies can be used to optimize the performance of storage operations, or queue servers to implement asynchronous operations on the server.

5. FEATURES

The web application must implement a set of groups of features, which will be described in this section. Their implementation must consider all the information about the business model described previously, the description of the groups of features (in this section: "5. Features"), the structure and description of the database tables and columns (in section: "8. Database"), as well as all the standards and good practices of the technologies and patterns involved. Students are free to decide on the user interface and usability, as well as on aspects of the business not explicitly described.

G1. VCARD CREATION AND DISMISSAL, AUTHENTICATION, PROFILE, SETTINGS

Any user (including anonymous users) can create a new vCard by specifying a phone number and a password (authentication credentials), his name, e-mail, and photo (optional) and a confirmation code (4 digits code that is used to confirm all debit transactions). The phone number must follow Portuguese rules for mobile numbers (9 digits starting with a 9 – e.g., 915345980) and the application must guarantee that the phone number is not already in use on an existing vCard. The vCard is always created with a balance of zero euros and a maximum limit ("max_debit") of 5000 euros (this limit can only be changed by the administrator).

Note: on a real system, the creation of a new vCard should be confirmed. One common pattern to implement a confirmation step would be to send a SMS message to the associated *phone with a confirmation code. However, in this project the confirmation step is not required – after the new vCard is created, the owner can start using it immediately.*

After the vCard is created the vCard owner can start using it. The credentials to authenticate as a vCard owner are the phone number and the password (both associated to the vCard). If someone owns more than one vCard, he must open a new independent session for each vCard – each session of the application is relative to a single vCard.

The application should allow the vCard owner to change his profile (name, e-mail and photo), the confirmation code and the password. Any changes to the confirmation code or password should be confirmed with the current password.

The vCard owner can configure the categories for the credit (income) and debit (expense) transactions. These categories are initialized with the default categories for the platform (categories copied from the default categories), but the owner is free to change or delete them or create new categories. If a category was already used to classify a transaction, it can only be deleted with a soft delete.

The owner can also dismiss (remove) the vCard from the platform, but only when the vCard balance is zero. Dismissing the card must be confirmed with the current password and confirmation code. If the vCard was never used before (has no transactions), it should be deleted from the database. If the vCard was used previously, then it should be removed with a soft delete.

G2. TRANSACTIONS

The vCard owner can use the web application to make any debit transaction (credit transactions are not allowed for the vCard owner), which always requires the value of the transaction, the type of payment (e.g., MBWAY; PayPal, vCard, etc.) and the reference of the payment (destination). The application must guarantee that the value is greater than zero (value is at least 0,01€), equal or less than the vCard balance (to guarantee that the balance is not negative) and equal or less than the maximum limit ("max_debit") for that vCard. The type of payment must be one of the currently supported types of the platform (list is defined on the table "payment_types" – do not include "soft deleted" rows). The application must also guarantee that the payment reference follows the validation rules for the chosen payment type.

Note: because DAD project uses a simplified version of the backend, there is no need to establish a "real" communication with any external financial entity. This means that validation should guarantee that all formats of the payment references are as correct as possible, but without making any confirmation that depends on the external entities. For *instance, if the payment type is MBway, the application should guarantee that the reference is a valid mobile phone number (with Portuguese format), but it does not need to guarantee that the number corresponds to a real MBway account.*

The exception are the vCard payments (internal transactions) where the application should verify if the number is a valid vCard number.

When a debit transaction is conducted by the owner, the application will automatically update the vCard balance as well as the date and datetime of the transaction. Also, all transactions will have 2 values (old balance and new balance) that should be calculated automatically – these values will help the vCard owner to better understand the balance fluctuations of his vCard.

In addition, the vCard owner has access to the vCard transactions history - a list of all transactions associated with that vCard - and from that list he can access detailed information of any transaction created previously.

Note: the history of transactions should be sorted by datetime, with the last transaction at the top of that list. Also, as the number of transactions can grow indefinitely, the application should be prepared to handle large sets of transactions (pagination, filters, etc.)

All transactions can have an optional description and category, which can be specified when the debit transaction is conducted or added latter to an existing transaction. The description and category of credit transactions can only be added after the transaction is created, because credit transactions of a specific vCard cannot be created by the owner of that vCard.

After creating a transaction, it cannot be deleted – it can be "soft deleted", but only when the vCard itself is "soft deleted". Also, the financial values and dates associated to any transaction are immutable – they cannot be changed. Only the description and category of a transaction can be modified after the transaction was created.

G3. ADMINISTRATION

Platform administrators will access the web application with different credentials (email and password) when compared to the vCard owners. These credentials (email and password), as well as the administrator's name can be view and modified in the administrator profile (changing the password should be confirmed with the current password).

Administrators can manually add credit transactions to any vCard on the platform - register transfers from external entities to a specific vCard.

Note: this feature only exists to simplify the application development using a simplified API. This will allow us to add positive balances to vCards without a transfer from another vCard or without a "real" transfer from an external entity (not supported by the simplified API).

Administrators will be able to manage other administrators – view the list of administrators and add or remove administrator accounts – and manage the vCards, where they can view, filter, and search the list of vCards. For each vCard they can access the phone number; name, e-mail and photo of the owner; balance and debit limit (max_debit) of the vCard and whether the vCard is blocked. Also, the administrator can block or unblock any vCard, change the debit limit (max_debit) of any vCard and delete vCards that have a balance of zero – if the vCard has no associated transactions, it is deleted from the database, if the vCard has associated transactions, then the vCard and all associated transactions are "soft deleted".

Note: when a vCard is soft deleted, it maintains all related data (vCard and transactions) on the database, but the application behaves as if the vCard does not exist.

When a vCard is blocked, the vCard cannot be associated to any new transaction (credit or debit) and the owner cannot access any application associated to that vCard.

Lastly, the administrators can configure the list of default debit and credit categories. This list of categories serves as a template for the categories of new vCards – when a new vCard is created the application copies the default categories to the list of categories associated to the vCard.

G4. STATISTICS

The application should provide to the vCard owner and administrators, statistical information visually (with graphs) and/or textually (with tables). vCard owners can only access financial information about their own vCard. Administrators can access global platform usage information or global (of all users) financial information – they cannot access any financial information about the individual vCards (except current balance) but they can access anonymized information about all vCards and all transactions - examples: current count of active vCards, sum of current vCard balances, count or sum of all transactions in a specific time frame, total transactions by type of payment, etc.

Students are responsible for the type and quality of information extracted from the platform and the way it is presented to the end users.

G5. AUTOMATIC REFRESH AND NOTIFICATIONS

The web application must support automatic refresh – it must be capable of changing the content automatically (without user intervention) if something "*external*" happens that affects the application state/content. This is especially important with the vCard balance and transactions history, to guarantee that when the vCard is credited (through a credit transaction), either by an internal transfer from another vCard or when the administrator registers a manual credit transaction, the application balance and history is immediately updated (only if the application is currently opened). Automatic refresh can also be implemented for other features – for instance, if the administrator deletes, blocks, or unblocks a vCard when it is being used, it is important that the application in use changes is content and behavior.

Complementing the automatic refresh, the application must also notify the user when something "*external*" happens – for example, if the vCard receives credit (a new credit transaction is conducted) from another vCard or from an external entity (when an administrator registers that credit transaction) the application should popup a notification message (in a notification window, panel, card or a simple "toast" message). The same should occur when the administrator deletes, blocks or unblocks the vCard (if it is being used in that moment), or any other situation that justifies the need to notify the user. These notifications should appear even when no automatic refresh is needed – for instance, if at a given moment the user is not viewing the vCard balance or transactions, he should nevertheless be notified when he receives money (credit transaction).

G6. CUSTOM FEATURES AND IMPLEMENTATION DETAILS

To distinguish the projects (students projects) from each other, students should add custom features that are not specified on the project statement or add implementation details that have one of the following characteristics: use a technology or technique not learnt during classes; optimize the performance; improve the architecture; improve the implementation structure or quality.

Students are responsible for specifying the custom features and implementation details.

Just as a reference, here is an example of a custom feature: enable any vCard owner to ask another vCard owner for money (request a payment of a specific value). The other owner should be notified, and if he accepts the request, the application could create both transactions (debit and credit) automatically. An example of an implementation detail could be to use a queue server or create a similar system, so that the application supports "delayed" notifications – for some type of notifications, if the vCard owner is not currently connected, show the notification only when the owner enters the application again.

6. CONSTRAINTS

Project’s **mandatory constraints** are the following:

C1. Application must use exclusively the Single Page Application (SPA) paradigm.

C2. Application must use the Vue.js framework for the client-side code.

C3. Applications’ web API (simplified version) must be implemented with the Laravel framework.

C4. Applications’ web socket server (if implemented) must be based on Node.js.

C5. Main relational database must use MySQL.

C6. Application must be published and accessible on the web through a desktop browser.

C7. The database of the published application must be seeded with data that simulates real life usage – both in size and content. Apply the provided seeder (Laravel seeder) - with the option "1-full" - to fill data on the published database.

7. NON-FUNCTIONAL REQUIREMENTS

Project’s non-functional requirements are the following:

NF1. The client application code and structure must follow the principles and good practices of the Vue.js framework.

NF2. The server-side code and structure of the web API (implemented in Laravel) must follow the principles and good practices of the Laravel framework.

NF3. The web API must follow the principles and good practices of a RESTful service.

NF4. Visual appearance and layout should be consistent through the entire application and adapted to the applications’ objective and usage context.

NF5. All implemented features must be correctly integrated and work consistently throughout the application.

NF6. Application should have the best usability possible. This implies that information and available operations are clearly presented to the user, and that user interaction is very simple, uses standard procedures, and requires a minimum of interaction from the user to obtain the required outcomes.

NF7. Data inputted by the user should always be validated on the client and on the server, according to rules extrapolated from the business model, database, and groups of features.

NF8. Application should be reliable and have an optimized performance. For instance, performance can be improved by applying a mix of techniques to minimize internet bandwidth (reduce the number of HTTP Requests and the size of HTTP Responses) and client memory (limiting the size of datasets on the client) while maximizing the performance of data navigation and filtering on the client (if data is already on the client, navigation and filtering data can be very fast using JavaScript code on the client). Other techniques can be applied to the Vue.js code to improve the performance on the client, and to the Laravel and MySQL code to improve the performance on the server.

NF9. Application should be as secure as possible and it should guarantee the protection of users' data and privacy, ensuring that no unauthorized user can view or change any data or document it should not have access to. Authentication and authorization must follow the principles and good practices for authentication and authorization of web APIs.

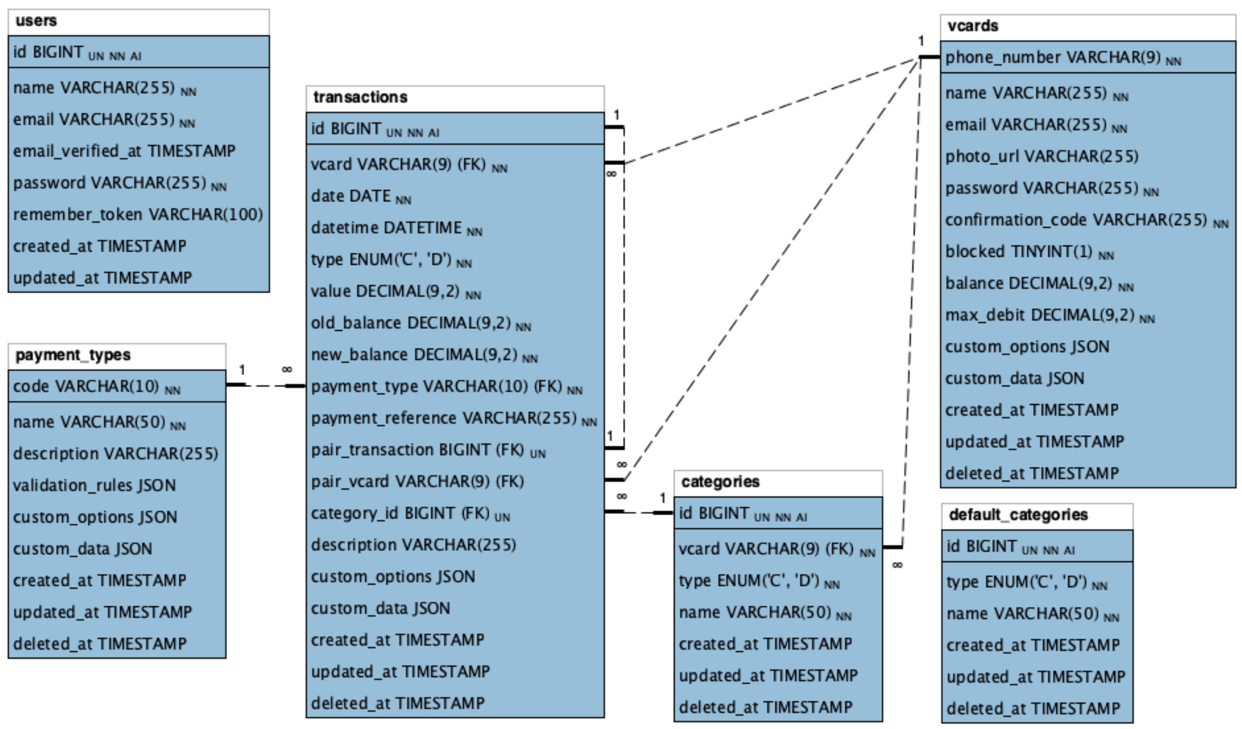
NF10. Performance of the Web Socket full duplex communication should also be optimized. This implies minimizing the size of the messages and sending messages exclusively to the clients that really require it (avoid broadcasting messages to all clients if they do not need to receive them).

8. DATABASE

The main relational database must use a MySQL server (it can be complemented by any Non-SQL database, created by the students, if they so decide). The **database structure is provided** (as a Laravel Migration) and the tables structure cannot be modified. However, students can add tables to the database (as long as their inclusion is properly justified). A **database seeder is also provided**, which can be used to preload the database with data that simulates the application usage through several months.

Note: The preloaded data of the database (after using the database seeder) fills all **passwords** & **confirmation codes** with: "**1234**" – the database seeder stores the hash for that value ("1234") using Laravel algorithms.

The database structure is as follows:



TABLES AND COLUMNS

The structure of tables and columns (for instance, foreign keys, data types of columns, whether columns are required or not) and the description presented here are relevant to the features of the application and validation of data entered by users.

USERS

Table with all administrator users. The credentials for these users are different from the credentials to access vCards.

- "**id**" – primary key – automatic integer.
- "**name**" – the name of the user.
- "**email**" – e-mail of the user. The e-mail must be unique, and it serves also as authentication credential for the administrators' (together with the password).
- "**password**" – *hash* for the users' password.

Following columns are internal to Laravel – typical applications will not use them directly:

- *"**email_verified_at**" (optional) – date when the user's e-mail was verified.*
- *"**remember_token**" (optional) – to implement the "remember me" feature*
- *"**created_at**" (optional) – date and time when the row was created*
- *"**updated_at**" (optional) – date and time when the row was last changed*

DEFAULT_CATEGORIES

List of default categories for the classification of credit or debit transactions. This list will serve as a template for the categories associated to a new vCard – when a vCard is created, the application will create a set of categories associated to the new vCard that are a copy of the current list of default categories.

- "**id**" – primary key – automatic integer
- "**type**" – type of transaction ("C"= credit; "D"=debit) that this category will classify
- "**name**" – name of the category (what the end-user will see)

Following columns are internal to Laravel – typical applications will not use them directly:

- ***"created_at"*** *(optional) – date and time when the row was created*
- ***"updated_at"*** *(optional) – date and time when the row was last changed*
- ***"deleted_at"*** *(optional) – date and time when the row was "soft deleted"*

CATEGORIES

List of categories for the classification of credit or debit transactions, associated to a specific vCard. Each vCard has its own set of categories. This list of categories is initialized (when the vCard is created) from the categories on the "default_categories" table, but afterwards, the vCard owner is free to organize the categories as he pleases (including removing or adding new categories).

- "**id**" – primary key – automatic integer.
- "**vcard**" – foreign key - the associated vcard (the phone number of the associated vcard)
- "**type**" – type of transaction ("C"= credit; "D"=debit) that this category will classify
- "**name**" – name of the category (what the end-user will see)

Following columns are internal to Laravel – typical applications will not use them directly:

- ***"created_at"*** *(optional) – date and time when the row was created*
- ***"updated_at"*** *(optional) – date and time when the row was last changed*
- ***"deleted_at"*** *(optional) – date and time when the row was "soft deleted"*

VCARDS

Table with all vCards of the platform, including deleted (with soft deletes) vCards.

- "**phone_number**" – primary key – the phone number identifies the vcard. All phone numbers have exactly 9 digits (format for Portuguese phone numbers).
- "**name**" – name of the vCard owner.
- "**email**" – email of the vCard owner.

- "**photo_url**" – (optional) photo of the vCard owner - relative url for the image of the photo.
- "**password**" – *hash* for the vCard password. The password is used as a part of the credential (phone number + password) for the application authentication (access to a specific vCard).
- "**confirmation_code**" – *hash* for the vCard confirmation code. The confirmation code has 4 digits and is used to confirm all transactions executed from the vCard.
- "**blocked**" – Indicates whether the vCard is blocked or not. Default value (when the vCard is created) is false, but the administrators can block (blocked = true) any vCard. When the vCard is blocked, the vCard cannot be associated to any transaction (credit or debit) and the owner cannot access any application associated to that vCard.
- "**balance**" – current balance (amount of money) of the card. Value cannot be negative (value is always >= 0). Value of the balance is always zero when the vCard is created.
- "**max_debit**" – the maximum value for any debit transaction of the vCard. The default (when the vCard is created) is 5000€. Only the administrators can change this value.
- "**custom_options**" – json data – this column can be used to add any custom option (configuration or other types of options) to the vCard. Students are free to ignore or add any type of data to this column (json data is not restricted to any structure).
- "**custom_data**" – json data – this column can be used to add any custom data to the vCard. Students are free to ignore or add any type of data to this column (json data is not restricted to any structure).

Following columns are internal to Laravel – typical applications will not use them directly:

- *"**created_at**" (optional) – date and time when the row was created*
- *"**updated_at**" (optional) – date and time when the row was last changed*
- *"**deleted_at**" (optional) – date and time when the row was "soft deleted". Note: if a vcard is "soft deleted", then all transactions associated to that vcard should also be "soft deleted".*

PAYMENT_TYPES

Table with all types of payments accepted by the platform. Transactions of the vCard are always associated to a payment, that defines where the money goes to (on a debit transaction) or where the money came from (on a credit transaction). Database seeder will fill this table with initially accepted payment types, namely: VCARD (internal payments within the vCard platform - transfer between 2 vCards); MBWAY; PAYPAL; IBAN (payments to or from banks); MB (payments with "multibanco") and VISA. The fact that these payment types are specified in a table, rather than being hard coded, allows the platform to change the accepted payment types over time.

> *Note: this table is not directly accessible through the web application – adding, changing, or removing a type of payment is done exclusively in the database.*

- "**code** " – primary key – code of the type of payment – identifies the type of payment throughout the platform.
- "**name**" – the name for the type of payment. What the end-user sees.
- "**description**" – text that describes the type of payment and how the payment references are specified. Examples: "vCard phone number"; "Bank Transfer - IBAN account code (2 letters and 23 digits)"; "PayPal email account"; etc.
- "**validation_rules**" – json data – the validation rules for the payment references of distinct payment types are completely different. For example: PayPal requires an email; vCard or MBway requires a phone number; IBAN requires a valid IBAN code; VISA requires a valid credit card number, MB requires an entity number (5 digits) and a reference (9 digits), etc. Since the payment types are not hard coded, validation rules must be configured within the table for the payment type - this column allows students to define dynamic validation rules adapted for each type of payment.

  Note: database seeder will leave this column empty. It is the responsibility of the students to define the structure and the data to use in this column.
- "**custom_options**" – json data – this column can be used to add any custom option (configuration or other types of options) to the type of payment. Students are free to ignore or add any type of data to this column (json data is not restricted to any structure).
- "**custom_data**" – json data – this column can be used to add any custom data to the type of payment. Students are free to ignore or add any type of data to this column (json data is not restricted to any structure).

Following columns are internal to Laravel – typical applications will not use them directly:

- *"**created_at**" (optional) – date and time when the row was created*
- *"**updated_at**" (optional) – date and time when the row was last changed*
- *"**deleted_at**" (optional) – date and time when the row was "soft deleted"*

TRANSACTIONS

Table with all the transactions (credit or debit transactions) associated to a vCard. This table will store the history of all the transactions associated to all vCards of the platform.

- "**id**" – primary key – automatic integer.
- "**vcard**" – foreign key - the vCard (the phone number of the vCard) that owns the transaction. Every transaction is associated to a specific vCard.
- "**date**" – the date of the transaction – filled automatically when the transaction is created. This column does not store the time of the transaction (just the date) allowing multiple transactions of the same day to be grouped together (transactions are indexed by vcard and date).

- "**datetime**" – the date and time of the transaction – filled automatically when the transaction is created.
- "**type**" – type of transaction ("C" is a credit transaction; "D" is a debit transaction).
- "**value**" – the value (€) of the transaction. If type of transaction is "C", the balance of the vCard increases; if type is "D", then the balance decreases. Only transactions with a "value" greater than zero (value is at least 0,01€) are allowed. Also, on a debit transaction, the value must be equal or less than the vCard balance (to guarantee that the balance is not negative) and equal or less than the maximum limit ("max_debit") for debits of the vCard.
- "**old_balance**" – the balance value (€) of the vCard before the transaction was conducted (the original balance).
- "**new_balance**" – the balance value (€) of the vCard after the transaction was conducted (the final balance). Although the "old_balance" and "new_balance" values are redundant, because they could be calculated from the history of all transactions, their presence in the transaction simplifies any data extraction that depends on the balance fluctuation through time.
- "**payment_type**" – the type of payment associated to the transaction. If the transaction is a debit, the payment refers to where the money was sent to (the destination of the money). If the transaction is a credit, the payment refers to where the money came from (the source of the money).
- "**payment_reference**" – the reference of the payment associated to the transaction. Note that different types of payment require references with different validation rules. For example: PayPal requires an email; vCard or MBway requires a phone number; IBAN requires a valid IBAN code (2 letters and 23 digits); VISA requires a valid credit card number (16 digits), MB_REF requires an entity number (5 digits) and a reference (9 digits), etc. – check table "payment_types" for more information.
- "**pair_transaction**" – when money is transferred between 2 vCards (payment type is vCard), the platform creates 2 transactions – the debit transaction in the original vCard (that sends the money) and the credit transaction in the destination vCard (that receives the money). These 2 transactions are mirrored (same value, one is a credit and the other is a debit) and we call them "paired transactions". This column references to the paired transaction (the other transaction/ the mirrored transaction) of the current transaction.
- "**pair_vcard**" – the vCard associated to the pair transaction. Although this column is redundant, because we could obtain the pair vCard from the pair transaction, its presence in the transaction table simplifies any data extraction about vCards internal transfers.
- "**category_id** " – (optional) the category of the transaction. This column is optional and only used if the vCard owner decides to classify the transaction (classification of the transaction is optional). The application must guarantee that a credit transaction only uses credit categories, and a debit transaction only uses debit categories.

- "**description** " – (optional) the description of the transaction (free text). This column is optional and only used if the vCard owner decides to describe the transaction.
- "**custom_options**" – json data – this column can be used to add any custom option (configuration or other types of options) to the transaction. Students are free to ignore or add any type of data to this column (json data is not restricted to any structure).
- "**custom_data**" – json data – this column can be used to add any custom data to the transaction. Students are free to ignore or add any type of data to this column (json data is not restricted to any structure).

Following columns are internal to Laravel – typical applications will not use them directly:

- *"**created_at**" (optional) – date and time when the row was created*
- *"**updated_at**" (optional) – date and time when the row was last changed*
- *"**deleted_at**" (optional) – date and time when the row was "soft deleted". Note that transactions should only be "soft deleted" if the associated vcard is also "soft deleted" - if a vcard is "soft deleted", then all transactions associated to that vcard should also be "soft deleted".*

9. DELIVERY AND PUBLISHING

The application must be published in the provided virtual machine and accessible through ESTG intranet via a desktop browser (tests will be made on google chrome). **<u>The functional evaluation will be made exclusively in the published application, so publication is not optional.</u>**

When publishing the project, all features and configurations should be as close as possible to the application final production stage. Publishing should include all performance optimizations and must use data that **<u>simulates real life application usage</u>** (apply the provided database seeder and select option: "[1] full").

Project delivery must include 2 files (code and report) per group and 1 individual report file per student. For example, if the group has 4 students, 6 files must be delivered (2 files related to the group and 4 individual report files) – one student delivers 3 files (2 group files and his own individual report), and the remaining 3 students deliver only their own individual reports. Files:

- prj_GG.zip – zip file that includes a copy of all the project’s folders and files (source code and other files), except the folders “vendor”, “node_modules” and ".git".
- grp_report_GG.xlsx – the group report file - includes group elements identification and information about the project implementation. The model for the report will be provided.
- individual_report_NNNNNNN.xlsx – the individual report file - includes self-assessment and peer review. The model for the report will be provided.

Note: replace GG with your group number and NNNNNNN with your student number

10. EVALUATION

The evaluation of the project will consider the compliance with the mandatory constraints (C1… C7), Groups of features (G1 … G6) and Non-Functional Requirements (NF1 … NF10).

Mandatory Constraints (C1…C7)

These are mandatory constraints. If the project does not comply with all these constraints (including application publishing), then it will have a **classification of zero**.

Groups of features (G1…G6)

For the classification of each group of features, it will be considered the compliance with the business model, the usability, the reliability and the integration with the application, the quality and structure of the project's source code to implement it, as well as the compliance of non-functional requirements applicable to the user story.

The classification of all groups of features is valued **70%** of the project. Individual values for each group of features are specified on the table with the weight of all criteria.

Non-Functional Requirements (NFR1…NFR10)

Non-functional requirements are valued **30%** of the total classification of the project. These requirements are evaluated as a whole (there is no individual classification for each non-functional requirement) through the entire application. The evaluation of these requirements is made in parallel with the evaluation of the functional requirements, by analyzing the code and structure of the project and by testing (manually or automatically) chosen Web API endpoints.

Weight of all evaluation criteria:

	Groups of features	**Weight**
G1	vCard creation and dismissal, authentication, profile, settings	**10%**
G2	Transactions	**15%**
G3	Administration	**15%**
G4	Statistics	**10%**
G5	Automatic refresh and notifications	**10%**
G6	Custom features and implementation details	**10%**
NFR	All Non-Function Requirements	**30%**